短納期型開発プロジェクトのためのテスト手法「FaRSeT（Flexible and Rapid Software Test）」の適用と効果

Effort of application of test method "FaRSeT (Flexible and Rapid Software Test)" for short-term delivery development project

日本ナレッジ株式会社　V&D 事業部　札幌事業所
Nihon Knowledge Co., Ltd. V&D Division Sapporo Office
○上田　和樹　丹場　順次[1)]　工藤　修悟[2)]
○Kazuki Ueda　Junji Tanba[1)]　Shugo Kudo[2)]

Abstract
In the short-term delivery development project, scoping, development and testings are conducted concurrently. However, this process raises the following issues: change to the scoping resulting in revisiting and reviewing the test case itself and also adversely impacting on the project timeframe and efficiency.
To respond to these specification changes occured during this project development process, the new process called FaRSeT (Flexible and Rapid Software Test) has been developed. This report articulates this new FaRSeT process.

1. はじめに

1.1. 背景

ソフトウェア開発のテストフェーズをアウトソーシング[1]として受託する場合には、最初にテスト要件・規模・スケジュールを発注元と合意する必要があるため、「テスト内容を事前に計画して合意する」「テストに必要なドキュメントが揃った状態でテストプロセスを開始する」といった、いわゆるウォーターフォールプロセスのテストを採用することが多かった。

だが、近年は開発開始から納品までの工期が非常に短い「短納期型開発プロジェクト[2]」に対する発注元からのテスト依頼が増えており、短納期ゆえに要件定義・設計・開発・テストといった開発プロセスが並行で実施される状況が見受けられる。

1.2. 既存の課題と問題

開発プロセスの並行は仕様変更の多発を招くことが多い。仕様変更の多いプロジェクトにウォーターフォールプロセスでのテストを実施すると、テストチームでは仕様変更による手順記述式テスト（以下、スクリプトテスト[3]）の記述修正が必要になるため、テスト工数が増える。さらには仕様変更に伴う開発スケジュールの変動により、テスト実行スケジュールがテスト対象プログラムの開発完了のタイミングに大きく影響される。そのため、計画通りのテストができないなどの非効率的な作業によりテスト工数が増えることが問題となる。

こういった仕様変更に伴う非効率的なテストを実施せざるを得ない状況の、本来の根本解決は「適切な納期の確保」や「開発プロセスの遵守」等であるが、プロジェクトの制限事項のため、これらの根本解決は実際の現場では難しい場合がある。よって、本稿では根本解決が難しい状況下での問題解決事例について述べる。

日本ナレッジ株式会社　V&D 事業部　札幌事業所
Nihon Knowledge Co., Ltd. V&D Division Sapporo Office
北海道札幌市中央区北一条西 3-3 敷島プラザビル 7F　Tel: 011-207-5090　e-mail:ueda@know-net.co.jp
Shikishima Plaza Building 7F, Kita 1-jo Nishi 3-3, Chuo-cut, Sapporo, Hokkaido, Japan
1) 日本ナレッジ株式会社　V&D 事業部　札幌事業所
Nihon Knowledge Co., Ltd. V&D Division Sapporo Office
2) 日本ナレッジ株式会社　V&D 事業部　札幌事業所
Nihon Knowledge Co., Ltd. V&D Division Sapporo Office
【キーワード：】探索的テスト　マインドマップ　短納期　品質特性

なお、当事例は以下のようなシステムに対するテストプロジェクトでの効果を確認している。

- 業務系システムのリプレースなど、システムが達成しようとする業務要件（利用者がシステムを使って達成したい業務処理やシナリオ）が概ね確定しており、業務要件とシステム要件（業務要件を達成するための UI や要件）の合致性が重要なシステム。
- 回避手段がある細かい不具合よりも、運用にかかわるような重大な不具合の検出を優先させることが要求されるシステム。

2. 問題点解決への施策

2.1. 施策概要

仕様変更に伴うテストの効率化のため、仕様変更の多い短納期プロジェクトに対応できるようなフレキシブルでスピード感のあるテスト設計及びテスト実施手法として、マインドマップ[4]によるテスト分析を活用した探索的テスト[5]を適用した。この手法は「FaRSeT（Flexible and Rapid Software Test）／読み：ファルセット」とチーム内で呼んでおり、段階的にブラッシュアップを重ねながら幾つかのプロジェクトに対して適用してきた。当手法を以降は「FaRSeT」と記述する。

FaRSeT では、仕様変更によるテストケースの修正を最小限にするために、まずは仕様変更の発生度合いを以下のように分類した。

- システム要件　⇒　仕様変更が多い要素
- 業務要件　　⇒　仕様変更が少ない要素

この分類により、仕様変更が多い要素に対しては抽象度の高いテスト分析及び設計を行い、仕様変更が少ない要素に対しては詳細なテスト分析及び設計を行うなど、仕様変更の度合いによってテスト設計及び分析のアプローチを変えることができる。具体的には、変更が多い要素（システム要件）に対してはテストスクリプトを記載しない探索的テストを採用して、頻繁に発生する仕様変更への対応を容易にした。また、仕様変更が少ない要素（業務要件）については、探索的テスト実施前にマインドマップを活用して業務分析を行った。そして、開発スケジュールの変更に応じた最善のテストを行うため、「探索的テストマトリクス」を活用して品質状況の把握と関係者との合意ができるようにした。

FaRSeT には、テスト設計とテスト実施についてのそれぞれの施策がある。

2.2. テスト設計時の施策

仕様変更によるテストケース修正の影響を少なくするために、テスト設計のためのテスト分析については仕様変更の度合いが少ない要素である「業務要件」の分析を中心に行うこととした。これは、「運用に関わる不具合の検出」の優先度が高いことも理由である。そして、仕様変更が多い要素についてはテスト手順を詳細に記述するスクリプトテストは採用せず、テスト設計とテスト実施を同時に行う「探索的テスト」（手順の非記述式テスト）を採用することにした。ただしアドホックテスト[6]とは異なる効果の高い探索的テストを実施するためには、以下の要件を満たす必要があると考えた。

- 探索的テストはテスト目的を事前に設定する必要があること。
- テスト担当者が、システムの想定利用状況と達成したい業務要件を十分に理解していること。
- テスト担当者がテスト対象システムを十分に理解していること。

そこで、上記の必要条件を満たすために、以下の施策を行う。

2.2.1. マインドマップによる分析

短納期開発において、テストのための業務要件の分析を行う際に十分な開発ドキュメントが揃っていないことが多く、開発側とテストチームが業務要件の不明点の確認などを行うことを目的として、視覚的効果と共有性が高い「マインドマップ」を活用することとした。業務要件の分析のために、図１のようなマインドマップを作成することにより「利用者分析（想定ユーザー）」「業務シチュエーション分析（考えられる業務処理やシナリオ）」「システム分析」を行い、探索的テストの目的（以降：テストチャータ[7]）の一つとすることとした。探索的テストを実行する際は、このマインドマップを必ず参照する。たとえば、業務シチュエーション分析の重要な部分（図 1）

での赤字箇所をテストチャータに組み込むことにより、探索的テストで実施するようにする。また、開発及びテストチーム内など利害関係者内でマインドマップのレビューを十分に行い、内容の合意を行うことでテストの漏れを減らすことを目指す。

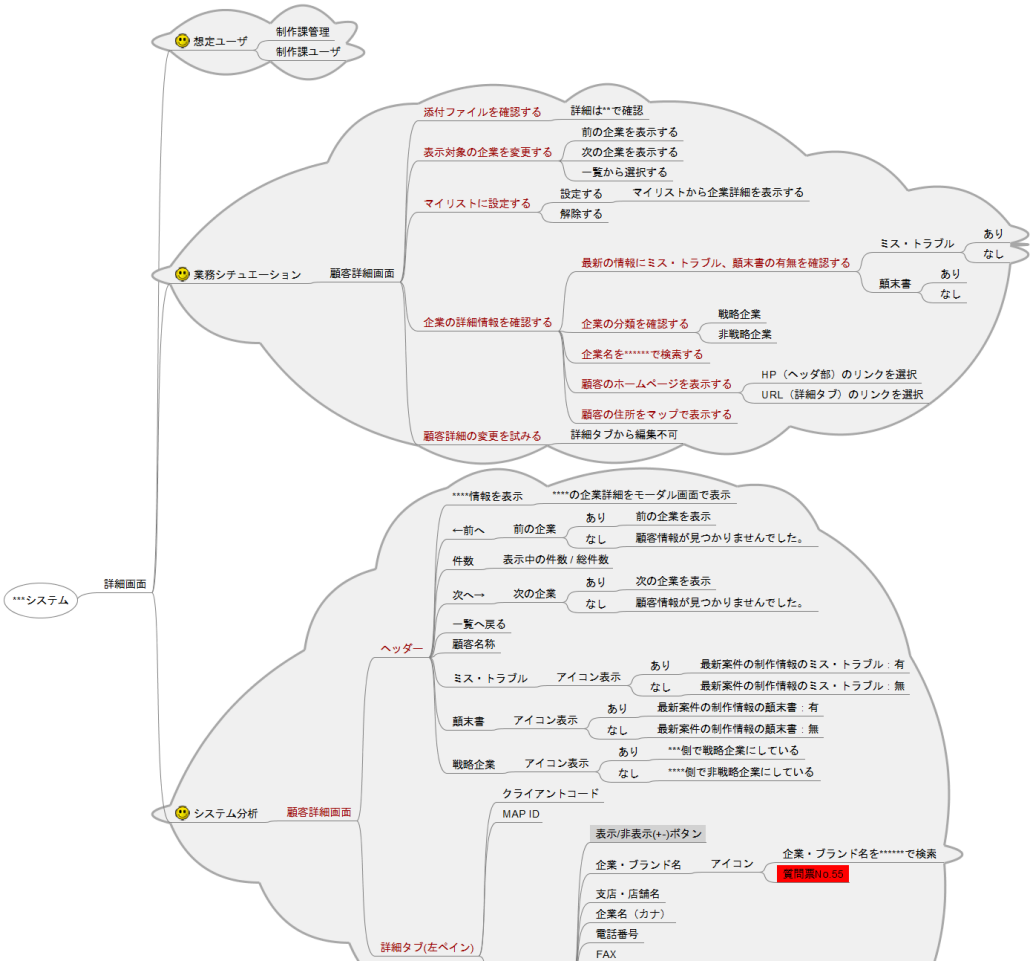

図 1. 業務分析マインドマップの例

2.2.2. テストチャータの作成

探索的テストをより効果的に行う工夫として、テストチャータの作成には、非機能を含めた広い観点でテストを行うために JIS X 25010:2013 [8]の「品質モデル」を活用し、品質モデルに定義された各品質特性からブレークダウンしてテストチャータを詳細化した（表 1)。

表 1. テストチャータの設計方法

品質特性		設計方法	
機能適合性	機能完全性	マインドマップによる分析と設計（図１）	想定ユーザー・業務シチュエーション分析
	機能適切性		
	機能正確性		システム分析
性能効率性		汎用テスト観点を使用（図２）	
互換性			
使用性			
信頼性			
セキュリティ			
保守性			
移植性			

機能適合性については、章 2.2.1 でのマインドマップに基づく分析にて詳細化していき、その他の品質特性についてはシステムのドメイン（たとえば「Web での業務系システムドメイン」など）ごとに、過去に行ったテスト実施結果のノウハウを汎用化してまとめた「汎用テスト観点」（図 2）をもとにテーラリングしてテストチャータを作成した（図 3）。テストチャータを探索的テストで利用することにより、テスト実施者にテストの気づきを与えることができる。また、マインドマップ作成時に浮かび上がった気になる箇所や疑問点なども盛り込みテスト実施時のヒントとした。

品質特性	副特性	概要	テスト観点
機能適合性	機能完全性	要件	・機能要件を満たしているか ユーザが求める機能が全てあるか 金融系、公共系、物流系、製造系などシステム特性に照らし合わせて適切な実装か ユーザの利用が想定される環境(PC、スマートフォン、iPadなど)で使えるか 業務で扱うデータを投入して使えるか 営業、オペレータなど関係部署を想定して各アクターでのシステム利用が可能か ・非機能要件を満たしているか インフラ、セキュリティ要求を満たしているか ・業務の利用目的を達成できるか
機能適合性	機能正確性	ブラウザの複数立上げ	・ブラウザの複数立上げ(Ctrl＋Nキー押下)時の操作に問題はないか
機能適合性	機能正確性	操作中に別サイトへ移動	・操作を途中で止めてサイトを去る場合に取り消されるか
機能適合性	機能正確性	データの整合性	・データ追加/更新/削除後、予期しないデータの損失はないか ・顧客名と属性情報(住所や電話番号)が合致しているか ・一覧⇔明細、メイン⇔詳細等、画面間でデータが正しく引き継がれるか ・同一画面内のヘッダーと詳細でデータの整合性が取れているか ・更新処理後、複数テーブルにある同一情報の整合性が取れているか ・顧客データがバージョン管理されている場合、最新のデータが表示されるか ・異なるアクセス権・ユーザーで登録・削除・更新した場合、それぞれの値が正しく表示されるか
機能適合性	機能正確性	名寄せ	・同一法人・同一人物の複数情報が一元的に取り扱われているか 例)平仮名⇔片仮名、片仮名⇔アルファベット、全角⇔半角、数字⇔漢数字 例)正式名称⇔省略名称 北1条西3丁目⇔北1条西3、タリーズコーヒー⇔タリーズ
機能適合性	機能正確性	ノーデータ	・0件に対応したメッセージや表示が正しく行われるか ・データが0件の場合、ボタンが非活性になるか ・複数のテーブルから取得してくる際、いずれかのテーブルのデータ0件だった場合にシステムエラーとならないか
機能適合性	機能正確性	DB同時アクセス	・画面表示後に他ユーザーが該当データを削除し、その後にデータを更新しても問題ないか ・ひとつのテーブルを複数のPRGから利用した場合、データの整合性に問題ないか
互換性	共存性	他システム同時動作	・他システムを起動しているとき、同時にテスト対象PRGが動作できるか クライアントではファイアウォールに影響するセキュリティソフトなど
互換性	相互運用性	ファイル共有	・WindowsとiPad間でファイルの共有ができるか ・複数ユーザ間でファイルの編集ができるか ・他システムとのデータのやり取りはできるか ・他システムとの連携I/Fが用意されているか ・リアルタイムまたはバッチより適切に連携できるか
互換性	相互運用性	外部API	・外部が提供するAPIを使用している場合、連携動作(データの受け渡しなど)が正常に出来ているか Google Maps、グラフモジュール、Facebook等

図 2. 汎用テスト観点の例

観点No	品質特性	副特性	概要	テスト観点
1	機能適合性	機能完全性	要件	・マインドマップの「業務シチュエーション」の機能要件が満たしているか ・ユーザの利用が想定される環境(PC、スマートフォン、iPadなど)で使えるか ・業務で扱うデータを投入して使えるか
2	機能適合性	機能正確性	画面遷移	・マインドマップの「システム分析」のメニュー階層上下関係に問題はないか →前後の画面遷移が行える(階層上がる/階層下がる) ・各種遷移ボタンを選択時、動作に問題はないか(完了、OK、キャンセル、「×」、次へ等) ・同一ウィンドウ、別ウィンドウが仕様通りか
			ブラウザの複数立上げ	・ブラウザの複数立上げ時の操作に問題はないか
			操作中に別サイトへ移動	・操作を途中で止めてサイトを去る場合に取り消されるか
			データの整合性	・データ追加/更新/削除後、予期しないデータの損失はないか ・一覧⇔明細、メイン⇔詳細等、画面間でデータが正しく引き継がれるか ・同一画面内のヘッダーと詳細でデータの整合性が取れているか ・異なるアクセス権・ユーザーで登録・削除・更新した場
			ノーデータ	・0件に対応したメッセージや表示が正しく行われるか ・データが0件の場合、ボタンが非活性になるか
			DB同時アクセス	・画面表示後に他ユーザーが該当データを削除し、その後にデータを更新しても問題ないか ・ひとつのテーブルを複数のPRGから利用した場合、データの整合性に問題ないか
4	互換性	共存性	他システム同時動作	・他システムを起動しているとき、同時にテスト対象PRGが動作できるか
5		相互運用性	ファイル共有	・複数ユーザ間でファイルの編集ができるか ・他システムとのデータのやり取りはできるか
			外部API	・Google カレンダー等外部が提供するAPIを使用している場合、データの受け渡しが出来ているか
7	使用性	習得性	マニュアル／ガイド	・画面上の案内文等ガイドの内容が理解しやすいか

図 3. テストチャータの例

2.3. テスト実施時の施策

2.3.1. 探索的テストマトリクスの作成

開発スケジュールの変更に応じて、テスト優先度の高い個所を特定して関係者同士で共有することを目的として、探索的テスト実施用のマトリクスを作成した（図 4）。マトリクスの横軸に 2.2 で作成したテストチャータを記載した。そして縦軸には、画面構成部品単位の粒度を目安として機能の一覧を記載した。

このマトリクスを作成することにより、以下が期待できる。

- 大きな仕様変更が生じた場合は、探索的テストマトリクスを修正するだけで済むので、スクリプトテストに比べて手戻り工数は少ない。
- 横軸のテストチャータは抽象度の高い記載を行っているので、仕様変更の影響を受けづらい。

2.3.2. 探索的テストマトリクスを用いたテストの実行

テストの実行は、図 4 の探索的テストマトリクスの交点（機能に対してのテストチャータ）を一つの単位（テストケース）として行った。探索的テストの実施結果はマトリクスの交点に不具合件数を記載していった。これにより、不具合の多い機能や、不具合の多い品質要件（テストチャータ）が認識できる。そして、この探索的テストマトリクスを開発チーム及びテストチームなどの利害関係者にて共有することで、品質状況の可視化と共有が行える。テストを実施する個所の選定には、プログラムの実装完了スケジュール（完成したものから実施する）、品質状況（不具合の多いテスト観点や機能を優先して実施する）、利用者のリスク、などを考慮して利害関係者と合意しながら優先度の高い個所（マトリクスの交点）から実施していった。この施策により、開発スケジュールが変更になった場合でも効率の良いテストが行えると考えた。

マトリクス上の凡例
：実施実績あり
：実施実績あり→不具合検出
※セル内の数字は：不具合チケット数

観点No					1	2	3	4	5	6	7	8	9	10	11	12
					機能適合性			互換性		使用性						セキュリティ
					機能完全性	機能正確性	機能適切性	共存性	相互運用性	適切度認識性	習得性	運用操作性	ユーザエラー防止性	ユーザインタフェース快美性	アクセシビリティ	機密性
					テスト分析した「業務シチュエーション」通りに動作するか	テスト分析した「システム」通りに動作するか	業務に対して的確な操作設計であるか	異なるシステムで各々処理が行われるか	外部モジュールとのデータ連携が行われるか	利用者がニーズを満たせるか判断できるか	利用方法を習得しやすいか	操作・表示の一貫性や、運用上の状態確認しやすいか	誤操作を防ぐ仕組みが実装されているか	画面デザインが見やすいか	幅広い範囲の人々が利用できるよう配慮されているか	適切な権限で操作・データの取得が行えるか
機能No	階層1	階層2	階層3	233	46	81	25	3	5	11	8	26	11	9	3	5
1	機能A	サブ機能		10	2	2	2				1			2		1
2			******ボタン	15	4	3	4					1	3			
3			*****	22	3	6	5		1	1	1	3	2			
4	機能B	サブ機能		6	1	2					1	1			1	
5			******一覧	6	3	2	1									
6			******一覧	5	1	3	1									
7			******一覧	0	0	0										
8	機能C	サブ機能		3	2	0						1				
9		サブ機能		2	1	1										
10		サブ機能		3	0	2						1				
11		サブ機能		3	0	0				2		1				
12		サブ機能		3	0	2										1
13		サブ機能		7	2	1	1			1			1			1
14	機能D	サブ機能		6	2	2						1		1		
15		サブ機能		1	0	1										
16		サブ機能		4	1	2						1				
17		サブ機能		4	1	2										1
18		サブ機能		1	0	1										
19		サブ機能		3	0	1	1				1					

図 4. 探索的テストマトリクスの例

3. 問題解決の測定

FaRSeT の効果を確認するために定量評価を行った結果が以下である。

3.1. 主効果（定量評価）

FaRSeT を適用したプロジェクトと適用しなかったプロジェクトを比較したものが表 2 である。

表 2. FaRSeT の適用の有無によるプロジェクト結果比較

プロジェクト名称	FaRSeT適用	テストケース数(A)	不具合検出数(B)	テスト工程の工数[人日](C)	ⅰ.テストケースに対する不具合の割合(B/A)	ⅱ.不具合検出に対する必要工数[人日](C/B)
プロジェクトA	あり	1,358	328	205	24.15%	0.6
プロジェクトB	あり	1,430	230	92	16.08%	0.4
プロジェクトC	なし	2,675	48	65	1.79%	1.4
プロジェクトD	なし	1,631	48	64	2.94%	1.3

FaRSeT を適用したプロジェクトに近いものを比較するため、過去の実施案件から WEB ベースでの業務系かつ、工数が 3～11 人月のプロジェクトを抽出したところ、4 つのテストプロジェクトが該当したため、それらを抽出して比較した。この表から以下が読み取ることができた。

- ⅰの結果より、FaRSeT を適用したプロジェクトは不具合の摘出効果が高い（不具合検出性の向上）。
- ⅱの結果より、FaRSeT を適用したプロジェクトは不具合検出に対する工数が少ない（工数の削減）。

また、同一プロジェクト内にて異なる 3 つのテスト手法での結果を比較した（表 3）。

- アドホックテスト
  - テスト経験のない担当者が、業務分析もテストチャータも用いることなく、本人の裁量によるテストを実施した。
- 探索的テスト
  - テスト経験のある担当者（5 年程度）による担当者が、テスト設計とテスト実施を同時に行いながら手順の非記述式テストを実施した。ただし、業務分析もテストチャータも用いない。
- FaRSeT
  - テスト経験のある担当者（5 年程度）による担当者が、業務分析とテストチャータを用いる FaRSeT にてテストを実施した。

FaRSeT は、他の手法（特に探索的テスト）に比べて不具合検出の必要工数が低いことが確認できた（工数の削減）。

また、表 4 はテスト手法別に検出された不具合の内訳を品質特性ごとに分析[9][10]したものである。業務要件とシステム要件の合致性は品質特性では「機能完全性」にあたる。表 4 の赤字部分から読み取れる通り、FaRSeT は他の 2 つの手法と比較して機能完全性に影響する不具合が検出できており、業務要件の欠陥が検出できていることがわかる。これは、FaRSeT においてはマインドマップで業務分析を行っていることが理由と推測している。また、FaRSeT は他の 2 つの手法よりも幅広い品質特性に影響する問題を検出できている。これは FaRSeT においては品質特性を用いたテストチャータを利用したことが理由と推測している。つまり FaRSeT のテスト手法は、検出したい重要な特性（機能完全性）の検出と、広い観点でのテスト実施を可能にすることができると考えられる（不具合検出性の向上）。

表 3. テスト手法による不具合検出数と工数

テスト手法	不具合検出数(D)	テスト工程の工数[人時](E)	不具合検出に対する必要工数[人日](E/D)
アドホックテスト	9	14	1.6
探索的テスト	10	27	2.7
FaRSeT	210	224	1.1

表 4. テスト手法による不具合検出内容の内訳

品質特性		アドホックテスト		探索的テスト		FaRSeT	
		件数	割合	件数	割合	件数	割合
機能適合性	機能正確性	4	40%	5	56%	74	35%
	機能完全性	−	−	−	−	42	20%
	機能適切性	−	−	2	22%	21	10%
使用性	運用操作性	3	30%	−	−	23	11%
	適切度認識性	−	−	1	11%	10	5%
	ユーザエラー防止性	1	10%	1	11%	10	5%
	習得性	−	−	−	−	8	4%
	ユーザインタフェース快美性	2	20%	−	−	7	3%
	アクセシビリティ	−	−	−	−	3	1%
互換性	相互運用性	−	−	−	−	4	2%
	共存性	−	−	−	−	3	1%
セキュリティ	機密保持性	−	−	−	−	5	2%
		10		9		210	

3.2. 副次的効果（定性評価）

- マインドマップでのテスト分析・設計工程で仕様の抜け漏れなどの指摘が行え、ドキュメントレビューを実施した場合と同様の効果があった（表 2 のプロジェクトAでは、テスト設計中の質問事項 61 件中、12 件が仕様漏れであった）。
- 設計書などのテスト実施のためのインプットドキュメントが不足していても、マインドマップ上でのレビュー時や探索的テスト時にシステムに対する学習効果がある。
- 仕様変更の影響を最小限に工期内でテストを進めることができた。
- 探索的テストマトリクスを全体共有しながら進めるため、関係者の納得を得ながら進めることができた。
- マインドマップでテスト設計を表現するために、特に発注元などの第三者にとってもテストする内容のイメージが湧きやすく、レビューが容易になった。

4. 結論

結論として、FaRSeT を短納期プロジェクトへ適用した場合に「テスト工数削減」と「不具合検出性の向上」の効果が認められた。

これにより FaRSeT の適用により以下の改善ができたと考える。

- 仕様変更による手順記述式テストの修正を要因とする、テスト工数増加の抑制。
- 仕様変更に伴う開発スケジュール変動を要因とする、非効率的な作業によるテスト工数増

加の抑制。

5. 適用の注意点

FaRSeT の適用については、以下を注意する必要がある。

- 開発及びテストチーム内で十分にテスト設計レビューを行う必要がある。
- 探索的テストマトリクスの縦軸の機能分析を適切な粒度感で行う必要がある。
- 探索的テスト実施担当者は、一定レベル以上の技術のメンバが実施する。
- 探索的テストの性質上、テスト結果の詳細なエビデンスが残りづらい。

6. 今後の検討

現在は業務システムを中心としたシステムでの適用が主だが、他の分野（組込系など）への適用も検討していきたい。

また、当手法だけではカバーできない領域(たとえば機能を網羅するテストなど)もあるので、別のテスト手法の併用も検討していきたい。そして、品質特性同士の関係性を整理することができれば、探索的テストマトリクスの実施個所の決定に対する重要な判断情報になるので、今後の課題としたい。

7. 参考文献

[1]IT 検証産業協会,https://www.ivia.or.jp/
[2]吉田千鶴,植木誠太郎,岡田ひろみ,櫻庭文吾,根本雄大,秋山浩一,上田和樹,喜多義弘,短納期でかつ小規模な開発プロジェクトにおけるテスト分析・設計導入による有効性の検証 ,ソフトウェア品質管理研究会,2017
[3]JSTQB,ソフトウェアテスト標準用語集 （日本語版） Version2.3.J02,p.29,2014
[4]池田暁,鈴木三紀夫,マインドマップから始めるソフトウェアテスト,技術評論社,2007
[5]高橋寿一,知識ゼロから学ぶソフトウェアテスト 【改訂版】,翔泳社、2013
[6]JSTQB,ソフトウェアテスト標準用語集 （日本語版） Version
2.3.J02(http://jstqb.jp/syllabus.html) ,p.11,2014
[7]JSTQB,ソフトウェアテスト標準用語集 （日本語版） Version
2.3.J02(http://jstqb.jp/syllabus.html) ,p.40,2014
[8]日本工業標準調査会,“システム及びソフトウェア製品の品質要求及び評価（SQuaRE）-システム及びソフトウェア品質モデル”,JIS X 25010,2013 年 6 月
[9]加藤大受,石川博,リカーリングビジネスを意識した問い合わせ分析のフィードバックプロセスの検討,ソフトウェア品質シンポジウム,2017
[10]上田和樹,丹場順次,加藤大受,サポート問合せ分析による、利用時品質向上及びシステム品質向上, ソフトウェア品質シンポジウム,2017